Step 1

Where are we?

Basic static HTML page.

What now?

Add basic Angular bindings to the page:

- Load angular.js

```
<script src="../js/angular.js"></script>
```

- Add ng-app directive

```
<body class="container" ng-app>
```

- bind inputs to models with ng-model directive

```
<div class="form-group">
<label for="customerName" class="control-label">Name</label>
<input type="text" id="customerName" class="form-control" ng-model="user.name">
</div>
<div class="form-group">
<label for="address" class="control-label">Address</label>
<input type="text" id="address" class="form-control" ng-model="user.address">
</div>
```

- display models with {{ ... }} interpolation bindings

```
<div class="well">
<a href="" class="pull-right">Change</a>
<strong>Deliver to:</strong><br>
{{ user.name }}<br>
{{ user.address }}
</div>
```

Step 2

Where are we?

Simple delivery details form.

What now?

Show and hide parts of the page dynamically:

- show the form deliveryForm.visible is true, using `ng-show`

```
<div class="row" ng-show="deliveryForm.visible">
```

- hide the delivery info display deliveryInfo.visible is true, using `ng-hide`

```
<div class="row" ng-hide="deliveryForm.visible">
```

- change deliveryInfo.visible when clicking "Change" and "Hide"

```
<a href="" class="pull-right" ng-click="deliveryForm.visible = true">Change</a>
...
<a href="" class="pull-right" ng-click="deliveryForm.visible = false">Hide</a>
```

Step 3

Where are we?

Dynamically visible delivery form.

What now?

Move the deliveryForm behaviour into a Controller:

- create an `app` module and a `FoodMeController` controller in `app.js`

```
angular.module('app', [])

.controller('FoodMeController', ['$scope', function($scope) {

}]);
```

- load the `app.js` file in the index.html page

```
<script src="app.js"></script>
```

- tell angular to load the app module

```
<body class="container" ng-app="app">
```

- attach the FoodMeController to the body of the view

```
<body class="container" ng-app="app" ng-controller="FoodMeController">
```

- initialize the scope with a deliveryForm object and a user object inside the `FoodMeController`

```
$scope.deliveryForm = {
  visible: true
};

$scope.user = {
  name: 'Jo Bloggs',
  address: '123, Some Place, Some Where'
};
```

- add `showDeliveryForm()` and `hideDeliveryForm()` helper methods to the `FoodMeController`

```
$scope.showDeliveryForm = function() {
  $scope.deliveryForm.visible = true;
};

$scope.hideDeliveryForm = function() {
  $scope.deliveryForm.visible = false;
};
```

Step 4

Where are we?

Simple structured app with code in app.js

What now?

Add a list of restaurants to choose from:

- Initialize a mock list of restaurants on the scope in the `FoodMeController`

```
$scope.restaurants = [
  {
    "price": 3,
    "rating": 3,
    "id": "esthers",
    "name": "Esther's German Saloon",
    "description": "German home-cooked meals and fifty-eight different beers on tap. To get more authentic, you'd need to be wearing lederhosen."
  },
  {
    "price": 4,
    "rating": 5,
    "id": "robatayaki",
    "name": "Robatayaki Hachi",
    "description": "Japanese food the way you like it. Fast, fresh, grilled."
  },
  {
    "price": 5,
    "rating": 4,
    "id": "bateaurouge",
    "name": "Le Bateau Rouge",
    "description": "Fine French dining in a romantic setting. From soupe à l'oignon to coq au vin, let our chef delight you with a local take on authentic favorites."
  }
];
```

- Bind the template to the list of restaurants using `ng-repeat` directive

```
<tr ng-repeat="restaurant in restaurants">
 <td class="description">
  <div class="media">
   <a class="pull-left">
    <img class="img-rounded" ng-src="../img/restaurants/{{restaurant.id}}.jpg">
   </a>
   <div class="media-body">
    <h4 class="media-heading">{{restaurant.name}}</h4>
    <p>{{restaurant.description}}</p>
   </div>
  </div>
 </td>
 <td class="rating">
  {{restaurant.rating}}
 </td>
 <td class="price">
  {{restaurant.price}}
 </td>
</tr>
```

Step 5

Where are we?

Displaying a list of restaurants from static mock data

What now?

Implement sorting of the restaurant list:

- Initialize `sortProperty` and `sortDirection` on the scope for sorting columns

```
$scope.sortProperty = 'name';
$scope.sortDirection = false;
```

- Add an `orderBy` filter to the `ng-repeat` using these properties:

```
ng-repeat="restaurant in restaurants | orderBy : sortProperty : sortDirection"
```

- Create helper methods in the controller, `sortBy(property)` and `getSortClass(property)`

```
$scope.sortBy = function(property) {
 if ( $scope.sortProperty === property ) {
   $scope.sortDirection = !$scope.sortDirection;
 } else {
   $scope.sortProperty = property;
   $scope.sortDirection = false;
 }
};
$scope.getSortClass = function(property) {
 if ( $scope.sortProperty === property ) {
   return 'glyphicon glyphicon-chevron-' + ($scope.sortDirection ? 'down' : 'up');
 }
};
```

- Convert the table headings to clickable anchors with `ng-click="sortBy('name')"` directives
- Display sort direction up/down markers using `ng-class="getSortClass('name')"` directives

```
<thead>
<tr>
 <th><a href ng-click="sortBy('name')">Name
   <span ng-class="getSortClass('name')"></span></a></th>
 <th><a href ng-click="sortBy('price')">Price
   <span ng-class="getSortClass('price')"></span></a></th>
 <th><a href ng-click="sortBy('rating')">Rating
   <span ng-class="getSortClass('rating')"></span></a></th>
</tr>
</thead>
```

- Link to the app.css stylesheet to fix column widths

```
<link rel="stylesheet" href="../css/app.css">
```

Step 6

Where are we?

Sortable list of restaurants

What now?

Create custom filter to display price and rating better:

- Create a custom `rating` filter in the `app` module - we must use `$sce.trustAsHtml` since we are generating HTML.

```
.filter('rating', ['$sce', function($sce) {
 return function(value, glyph) {
  var output = "";
  while(value>0) {
   output += '<span class="glyphicon glyphicon-' + glyph + '"></span>';
   value -= 1;
  }
  return $sce.trustAsHtml(output);
 };
}]);
```

- Use the filter in the price field, with the `ng-bind-html` directive

```
<td class="price" ng-bind-html="restaurant.price | rating : 'gbp'">
</td>
```

- Use the filter in the rating field, with the `ng-bind-html` directive

```
<td class="rating" ng-bind-html="restaurant.rating| rating : 'star'">
</td>
```

Step 7

Where are we?

Sortable restaurant list with basic delivery form

What now?

Add validation to delivery form:

- Load the `../js/angular-messages.js` file

```
<script src="../js/angular-messages.js"></script>
```

- Add the `ngMessages` modules as a dependency of our `app` module

```
angular.module('app', ['ngMessages'])
```

- Give the form a name (`deliveryForm`) so that it is attached to the scope
- Given the inputs names (`userName`, `userAddress`) so that they are accessible in the form object
- Add `required` and `ng-minlength="..."` validators to the inputs
- Update the classes on the `form-group` elements when the inputs are invalid

```
<div class="form-group" ng-class="{'has-error': deliveryForm.userName.$invalid}">
```

- Use `ng-messages` directive to display errors

```
<div ng-messages="deliveryForm.userName.$error">
  <div ng-message="required" class="alert alert-warning" role="alert">You must enter a name.</div>
  <div ng-message="minlength" class="alert alert-warning" role="alert">Your name must be at least 5 characters long.</div>
</div>
```

- Fix up the initial value of `deliveryForm.visible` inside an `$scope.$evalAsync()` call

```
$scope.$evalAsync('deliveryForm.visible = true');
```

Step 8

Where are we?

Sortable restaurant list with validated delivery form

What now?

Persist the delivery info in the local storage

- Create a new `localStorage` module in `localStorage.js`

```
angular.module('localStorage', [])
```

- Create a `localStorage' service to wrap the browser's localStorage object

```
.value('localStorage', window.localStorage)
```

- Create a `localStorageBinding` service that connects a property on the scope to the localStorage

```
.factory('localStorageBinding', ['localStorage', '$rootScope', function(localStorage,
$rootScope) {

 return function(key, defaultValue) {
  defaultValue = JSON.stringify(defaultValue || {});
  var value = JSON.parse(localStorage[key] || defaultValue);

  $rootScope.$watch(function() { return value; }, function() {
   localStorage[key] = JSON.stringify(value);
  }, true);

  return value;
 };
}])
```

- Load the new `localStorage.js` file

```
<script src="localStorage.js"></script>
```

- Add the new `localStorage` module as a dependency to our `app` module

```
angular.module('app', ['ngMessages', 'localStorage'])
```

- Inject the `localStorageBinding` service into the `FoodMeController`

```
.controller('FoodMeController', ['$scope', 'localStorageBinding',
      function($scope, localStorageBinding) {
```

- Bind the `user` object to the localStorage using the `localStorageBinding` service

```
$scope.user = localStorageBinding('foodMe/user', {
 name: 'Jo Bloggs',
 address: '123, Some Place, Some Where'
});
```

Step 9

Where are we?

Static mock restaurant list, with locally persisted delivery info

What now?

Load the restaurant data from a server

- Add the `$http` dependency to the `FoodmeController`

```
.controller('FoodMeController', ['$scope', 'localStorageBinding', '$http', function($scope,
localStorageBinding, $http) {
```

File system app and remote CORS enabled data server

- Replace the static restaurant data with a request to a REST service (https://foodme.firebaseio.com/.json)

```
$http.get('https://foodme.firebaseio.com/.json').then(function(response) {
  $scope.restaurants = response.data;
});
```

Locally hosted http server app and data

- Install a local webserver

```
npm install -g http-server
```

- Start the server in the root of the project

```
cd foodme-intro
http-server
```

- Browse to the application via this server: `http://localhost:8080/step-9`
- Now you can get the restaurant data from the local server

```
$http.get('../data/restaurants.json').then(function(response) {
  $scope.restaurants = response.data;
});
```

Step 10

Where are we?

List of restaurants loaded from a REST service

What now?

Filter the restaurants by price and rating:

- Add a new form to the left of the restaurant list

```
<div class="col-md-3">
 <form role="form" class="well" name="filterForm">
  <legend>Filter Restaurants</legend>
  <div class="form-group">
   <label for="priceFilter" class="control-label">Price (no more than)</label>
   <input type="number" id="priceFilter" name="priceFilter" class="form-control" ng-
model="filters.price">
  </div>
  <div class="form-group">
   <label for="ratingFilter" class="control-label">Rating (at least)</label>
   <input type="number" id="ratingFilter" name="ratingFilter" class="form-control" ng-
model="filters.rating">
  </div>
 </form>
</div>
```

- Initialize the filters to null on the scope

```
$scope.filters = { price: null, rating: null };
```

- Watch the price and rating values and filter the restaurant list accordingly

```
$scope.$watchGroup(['filters.price', 'filters.rating', 'restaurants'], function
filterRestaurants() {
 $scope.filteredRestaurants = [];
 angular.forEach($scope.restaurants, function(restaurant) {
  if ( ( !$scope.filters.rating || restaurant.rating >= $scope.filters.rating ) &&
      ( !$scope.filters.price || restaurant.price <= $scope.filters.price ) )
  {
   $scope.filteredRestaurants.push(restaurant);
  }
 });
});
```

- Change the `ng-repeat` directive to use the `filteredRestaurants

```
<tr ng-repeat="restaurant in filteredRestaurants | orderBy : sortProperty : sortDirection">
```

Step 11

Where are we?

List of restaurants retrieved from server, sortable and filterable

What now?

Add a cool rating directive for use in filtering:

- Create a new file `rating.js` containing a `rating` module

```
angular.module('rating', [])
```

- Load the `rating.js` file

```
<script src="rating.js"></script>
```

- Add the `rating` module as a dependency of our `app` module

```
angular.module('app', ['ngMessages', 'localStorage', 'rating'])
```

- Define a `fmRating` directive in the `rating` module

```
.directive('fmRating', function() {
 return {
  restrict: 'E',

  scope: {
   glyph: '@',
   rating: '='
  },

  link: function(scope, element, attrs) {

   scope.ratings = [1,2,3,4,5];

   scope.select = function(value) {
    scope.rating = value;
   };

   scope.isSelected = function(value) {
    return scope.rating >= value;
   };
  },

  template:
   '<ul class="fm-rating">' +
   '  <li ng-repeat="value in ratings" ng-click="select(value)" ng-class="{selected:
isSelected(value)}">' +
   '    <span class="glyphicon glyphicon-{{glyph}}"></span>' +
   '  </li>' +
   '</ul>' +
   '<a ng-click="rating = null">clear</a>'
 };
});
```

- Use this directive in the **Filter Restaurants** form instead of the input boxes

```
<fm-rating rating="filters.price" glyph="gbp"></fm-rating>
```

Step 12

Where are we?

Filterable, sortable restaurant list loaded from a REST service

What now?

Display the number of filtered restaurants:

- Add a binding to the length of the `filteredRestaurants` collection using `ng-pluralize` directive

```
<div class="alert alert-info" role="alert">
 <ng-pluralize
  count="filteredRestaurants.length"
  when="{'0'   : 'No restaurants found.',
       'one'  : 'Only 1 restaurant found!',
       'other': '{} restaurants found!'}">
 </ng-pluralize>
</div>
```

Step 13

Where are we?

Filterable, sortable restaurant list loaded from a REST service

What now?

Add transition animations to the delivery form:

- load `../js/angular-animate.js` file

```
<script src="../js/angular-animate.js"></script>
```

- Add a dependency on `ngAnimate` to the `app` module:

```
angular.module('app', ['ngMessages', 'ngAnimate', 'localStorage', 'rating'])
```

- add new class (`fade`) to the delivery form and delivery info elements to be animated

```
<div class="row fade" ng-show="deliveryForm.visible">
```